EXHIBIT C

The AdviceNet Project

Tuesday, March 11, 1997

Introduction

Computer reliability is like the weather: everybody talks about it, but nobody seems to do anything about it. The reason, perhaps. is that while we sometimes perceive reliability as a s single issue, problems with reliability come from a great many sources, and have an enormous variety of solutions.

In fact, many individual reliability problems have been solved, and more solutions are found each day. Hardware vendors replace faulty parts; software authors revise their work; and users themselves experiment with their machines to find ways to keep them working.

Therefore, we propose to attack the problem of reliability from a different angle — we will treat it not as a software-and-hardware problem, but as a communication problem. Rather than solving problems ourselves, we will provide a way to deliver solutions to the people who need them.

The State of the Art

We are not the first to tackle this communication problem. Most users communicate their solutions informally; many reach a wider audience through user groups and bulletin boards; some sell their advice in newsletters or magazine columns. Technical support workers are paid in part to distribute such advice; system administrators are paid in part to collect and sift through the advice others have made available.

But these existing pathways of communication have weaknesses we think we can improve upon. Informal communication reaches a limited audience; information is often corrupted as it spreads in this way. User groups, bulletin boards, newsletters, and advice columns require attention even when their advice is not pertinent. Technical support workers are expensive, and have no effective way to broadcast their advice to those who need it. System administrators are even more expensive, and have limited capacity to remember solutions to rare problems.

Our Solution

We propose to create an automated system which uses the internet to distribute advice. Our plan is to build software which establishes a subscription-based, point-to-point network which will deliver signed advice to users in a mixed machine- and human-readable format.

We envision a subscription-based system because we see the relation between an advisor and an end user as an ongoing one. Advisors not only create fresh advice from time to time, but also update and correct their previous advice. We want users to receive these updates with as little effort as possible.

We envision a point-to-point network because there are so many sources of advice that collecting it all would be a monumental task, and because these are so many users that distributing it all would be still more difficult. A point-to-point network distributes this work over many machines on the internet. Further, it gives us a first layer of screening and security: users will (we hope) not subscribe to advice sources which they find irrelevant or untrustworthy. Finally, it allows for privacy: advice can be provided on a part of the internet which is not globally accessible.

We will provide a second layer of security by insisting that advice be signed. The principal defense a user has against bad advice is knowledge of the source: advice from a trusted source is usually good advice. We will build upon that trust by making sure the source is known, and verifying the source cryptographically.

We want at least some portions of the advice to be machine-readable because we

want to automatically filter advice for relevancy. We imagine that most advice will concern particular hardware or software configurations, and that most advice will be irrelevant to most users. A significant part of the value we can provide is in locating that portion of the available advice which is relevant. In the ideal situation, the actions advised will also be machine-executable, so that the user will only be called upon to accept or reject the advice.

On the other hand, we realize that neither the conditions under which advice applies nor the advised actions can always be handled exclusively by the computer; even if they could, a user should have the chance to understand the advice before accepting or rejecting it. Therefore it is essential that advice be provided in a human-readable format.

Delivery

We intend to use HTTP as the primary delivery mechanism for advice. Each user will have a list of URIs to be searched for advice; the user's machine will periodically check those locations for new advice or updates to old advice. The URI will point to a table of available advice, which will contain the URIs and dates for advice files and other advice tables. This nested approach should allow for a relatively small transaction when small changes are made by an advice supplier.

While the user ought to be able to subscribe to an advice site by typing in its URI, we imagine that the subscription will more typically be created by an internet helper program which interprets the URI by adding it to the subscription list. This will make it easy to advertise advice sites on the web, by email, or through other advice sites. Also, we will want to provide a mechanism by which a program's installer could subscribe the user to the program's official advice site.

We imagine that the program will collect advice and deposit it in a database on the user's machine. In doing so, it may make a judgment as to the eventual pertinence of the advice — there's no reason for a user's machine to download or remember advice about products the user doesn't own. But advice about things which work now but could fail later should be kept.

Checking

Some advice which is not relevant when it is first loaded into the machine may become relevant with the passage of time. In particular, advice which describes a properly-installed piece of software may be triggered when a file is moved or modified. To handle this possibility, the program will need to continually compare the state of the machine to the advice in the database. When the conditions attached to a piece of advice are fulfilled, it will report that advice to the user.

The heart of this comparison seems to be a sort of outline-based pattern matching. We imagine that a system can be described as an outline: it has various pieces of hardware and software packages; the packages have folders and files; the devices, folders, and files have various attributes. Advice will come with patterns attached to it; when the system's outline matches the pattern, the advice is relevant.

While the description above covers the intended result, it's not the algorithm we want to use. For efficiency we'll only generate those parts of the system description which play a part in the pattern matching.

Reporting

When a piece of advice becomes relevant, we'll prepare a report for the user, containing, or at least summarizing, all of the relevant advice. Our current feeling is that the report will be built from passages of MIME content such as text or HTML, and presented in a manner similar to email, giving the user an "inbox" of relevant advice. We'll allow advice authors to categorize advice, which will then be sorted automatically into folders within the inbox.

To simplify the presentation of solutions to problems, we will allow advisories to be adorned with buttons which trigger scripts written in some system-specific scripting language. Advisories written in HTML may contain links to web pages and forms, allowing the user to engage in an online dialog with a web server. We may also want to allow advisories to present a more complicated user interface written in Java.

Taking Action

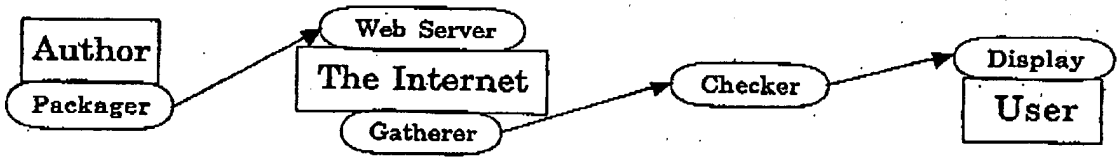
While the only action necessary in response to some advice is to instruct the user, in other cases we imagine that a full response could be available to the user at the click of a button. We want to provide advice authors the ability to execute scripts in the system's native scripting language, and to download and execute programs available on the net.

In addition, we expect to include utilities which will handle common advice actions, and won't have to be downloaded from the net. These actions include locating and updating files, changing advice subscriptions, and purging advice files from the user's database.

Examples

To solidify the way the system works, we'll follow several pieces of advice from the author through their eventual resolution. We'll start with a very basic piece of advice: some printers have been shipped with a defective part, and the user can ask for a replacement.

This advice follows a simple path from the author to the user:



To start the process, the author writes the advice, using a program we'll supply which will have a user interface similar to an email editor. Here's what the author will write:

```
Subject: Some LaserWriter IINT Printers are defective
Guard: exists printer where (model of it is "LaserWriter IINT")
Display-Path: Printer

LaserWriter IINT Printers with serial numbers between 123 and 456
are defective. Call (800)123-4567 for a replacement part.
```

The guard line describes a condition which must be met before this piece of advice will be considered relevant: in this case, there must be a LaserWriter IINT printer available to the system. The display-path line categorizes this advice, putting it into the folder "Printer" folder within the advice inbox.

The advice editor packages this advice by adding lines to the header:

```
From: user@host.com (The author)
Date: Tue, Mar 4, 1997 12:30 PST -0800
Subject: Some LaserWriter IINT Printers are defective
Guard: exists printer where (model of it is "LaserWriter IINT")
Display-Path: Printer
Signed: 895798579469658978932749865265 89
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

LaserWriter IINT Printers with serial numbers between 123 and 456
are defective. Call (800)123-4567 for a replacement part.
```

The only interesting line added here is the signature line. We want to encourage advice authors to digitally sign their advice, so that users will have greater confidence in the advice they receive.

Next, the author makes the advice available, by placing it in on a web site to which owners of these printers are expected to subscribe. In this case, an advice site would presumably be maintained by the company selling printers, and users would have either had a subscription installed with their printer software, or have found the advice subscription through the internet.

The user's machine, during a routine check for advice updates, will notice this advisory and download it. In the process, it will add a line indicating the source and time of arrival of the advice:

```
Received: from http://www.apple.com/PrinterAdvice.adv
     by localHost (123.234.123.234);
     Mon, Mar 10, 1997 14:30:16 PST -0800
```

Upon receipt, the advice is checked for relevance; that is, the guard lines on the

advisory are tested. If all the guard lines can be evaluated and are satisfied, the advisory will be listed in the display application. If not, it will be filed and rechecked periodically until it expires.

The advisory will wait in the user's advice inbox, filed under its display path "Printer" until either the user opens it or it becomes irrelevant. Eventually, the user will read it, decide how to respond to the advice, and throw it away.

The preceding example is of the simplest kind of advice, a plain text message. Now we'll move on to some more complicated pieces of advice.

Our next example is of a free minor update to an application. This piece of advice has two guard lines, both of which must be satisfied before the advice is considered relevant. It also suggests a response: an updater program is available on the web.

```
Subject: A new version of MicroPhone is available
Guard: version of application id 'DFBO' >= version "4.0"
Guard: version of application id 'DFBO' < version "4.0.3"
Display-Path: MicroPhone
Button: "Install"
Script: tell application "Downloader" to run
     "http://www.svcdudes.com/MicroPhone/updates/MicroPhone 4.0.3"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 59

Version 4.0.3 of MicroPhone is now available.  It fixes several
bugs, including blah, blah, and blah.  You may upgrade to this new
version for free.
```

When the display application shows this advice, it will put a button named "Install" next to the text of the message. If the user clicks this button, the script will run, downloading the updater and running it. The downloader application is one of the simple applications we will provide for executing standard kinds of advice.

Sometimes there's more than one way to respond to an advisory. An advice writer can include more than one MIME section in a single advisory to produce a list of

responses. Each section can have its own button and script; when the user chooses a button, the script is executed.

In this example, a file associated to the application has been delcted or moved from its proper location. Three options are presented: search for the file on local disks; download a replacement from the internet; or reinstall the file from the application disks.

```
Subject: MPToolbox is Missing
Guard: version of application id 'DFBO' >= version "3.0"
Guard: !exists file "MPToolbox"
        of parent directory of application id 'DFBO'
Display-Path: MicroPhone
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

The file "MPToolbox" is not in the MicroPhone folder. Without it,
some MicroPhone scripts will not operate correctly.

Button: "Search"
Script: tell application "Searcher" to launch
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

The file may have been moved to another place on the disk.  If so,
the problem can be solved by moving it back to the MicroPhone
folder.

Button: "Download"
Script: tell application "Downloader" to
    copy from "http://www.svcdudes.com/MicroPhone/files/MPToolbox"
    to parent directory of application id 'DFB0'
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

You can solve this problem by downloading "MPToolbox" from the
Software Ventures web site.

Button: "Reinstall"
```

```
Script: tell application "Finder" to
          open alias "MP Install 1:Install MicroPhone"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

This problem can be solved by reinstalling MicroPhone from disks.
```

While the advisory's window is open, the display application will closely watch the guard on the advisory; if the problem is fixed, the advisory will become irrelevant and the window will be closed. If the problem is not fixed, the user can choose another option.

Sometimes there are responses which are only appropriate to some users having a problem. Therefore, we will allow guards to be placed on sections of an advisory. Those sections will be shown only to users whose machines satisfy the guards.

In this example, an application has been modified; the advice detects the modification by using a checksum. The problem can be solved by reinstalling the program, but if the user has the virus checking application Disinfectant, the advisory suggests checking the application for viruses.

```
Subject: MicroPhone has been altered or damaged
Guard: version of application id 'DFB0' = version "4.0.2"
Guard: checksum of application id 'DFB0' != "AB76E48CF09B533F"
Display-Path: MicroPhone
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 59

The application "MicroPhone" has been altered or damaged.

Guard: exists application id 'D2CT'
Button: "Disinfect"
Script: tell application id 'D2CT'
                to open application id 'DFB0'
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 80

This problem may have been caused by a computer virus.  You can
check for known computer viruses by running "Disinfectant."
```

```
Button: "Reinstall"
Script: tell application "Finder" to
          open alias "MP Install 1:Install MicroPhone"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

This problem can be solved by reinstalling MicroPhone from the
original disks.
```

In this case, the first response is unlikely to solve the problem, and is not presented as a solution, but just as a suggestion. As before, the advisory window will stay open until the user dismisses it or the problem is fixed.

Sometimes, a plain text interface is insufficient to convey the content of an advisory, or a simple button is not enough of a user interface to provide a proper response to the advice. For these cases, we will offer HTML as an alternative to plain text. In the example below, an HTML form allows a user to order an upgrade to an application over the web. For those users who don't care to order over the web, a phone number is provided; if the computer can dial the phone, it offers to dial that number.

```
Subject: A new version of MicroPhone is available
Guard: version of application id 'DFBO' < version "4.0"
Display-Path: MicroPhone
MIME-Content-Type: TEXT/HTML
MIME-Content-Length: 276

<html>
Version 4.0.3 of MicroPhone is now available.  This new version
has cool features and is fully buzzword-enabled. You can upgrade
to this new version for a tidy sum.<p>
<form method=post
     url="shttp://www.svcdudes.com/cgi-bin/upgrade.cgi">
Credit card:
<select name="CardType">
     <option>American Express</option>
     <option>Discover</option>
     <option>Mastercard</option>
```

```
    <option>Visa</option>
</select><br>
Number: <input type=text name="CardNumber"><br>
Expiration date: <input type=text name="Expiration"><br>
<input type=submit name="Purchase">
</form></body></html>

Guard: exists extension "Telephony"
Button: "Dial"
Script: tell application "PhoneDialer" to call "(800)734-6727"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

You can also order Microphone by phone at (800)734-6727.

Guard: !exists extension "Telephony"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 87

You can also order Microphone by phone at (800)734-6727.
```

Our final example doesn't show off any further features of the system, but rather demonstrates that this system can be used in conjunction with a program's existing error-checking mechanisms to get users with problems in touch with technical support. In this example, a piece of advice looks for an error log, and suggests that the user deliver the log to the maintainers of the program:

```
Subject: MicroPhone reports unexpected errors
Display-Path: MicroPhone
Guard: exists file "MicroPhone Error Log"
       of parent directory of application id 'DFBO'
Button: "Send Report"
Script: tell application "Uploader" to submit
     file "MicroPhone Error Log"
        of parent directory of application id 'DFBO'
     to "http://www.svcdudes.com/cgi-bin/bugReporter.cgi"
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 59
```

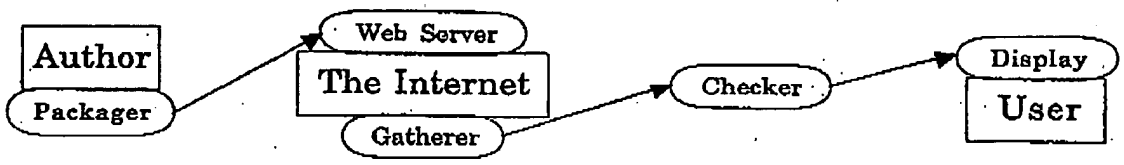
```
MicroPhone has encountered problems which were not fully
anticipated by the programmers. The program has compiled a report
of these problems. To help them fix these problems, please send
this report to Software Ventures.

Guard: size of file "MicroPhone Error Log"
       of parent directory of application id 'DFBO' > 100000
Button: "Delete"
Script: tell application "Finder" to move
     file "MicroPhone Error Log" to the trash folder
MIME-Content-Type: TEXT/ASCII
MIME-Content-Length: 59

The reports of MicroPhone's problems have grown to more than
100K.  If you do not wish to send these reports to Software
Ventures, you may throw them away.
```

The Central Parts

The basic path advice takes from its author to its recipient passes through five major pieces of software, four of which we'll write:



In the examples above, we've seen how these parts work together to deliver advice to the user. In the descriptions below, we'll see in more detail how the parts themselves will work. The descriptions are arranged in the order I expect they should be written, working away from the user.

The Display Application

The display application is where the user interacts with advice. Its main interface is similar to an email reader, presenting a list of advisories arranged by subject, date of reception, or date of relevancy. The user will have two three-way choices

for filtering the advice. The display can be set to show read, unread or both read and unread advisories, and can independently be set to show relevant, irrelevant, or both relevant and irrelevant advisories; the default will be to display all relevant advisories. The advisories will be further sorted by their display paths, producing a built-in hierarchy of folders.

The display application will also be able to open advice files directly, rather than as part of the subscription and system-monitoring mechanism. When the user opens an advice file directly, it will be displayed in a separate window from the inbox, but in other respects it will operate similarly.

When an advisory is opened, its signature and contents will be shown, including any buttons associated to its sections. When the application is set to show irrelevant advice, those sections which are irrelevant will be distinguished visually, and their buttons will be disabled.

While the display application is running, it will check continually for relevant advice becoming irrelevant. As it notices these changes, it will remove the advice from the list (if irrelevant advice is not being shown) and will close any displaying the advisory.

The display application will also allow the user to add or remove advice from the system, check for updates, or to change advice subscriptions. In particular, when viewing an advisory, the user should be able to directly throw it away, check it for updates, or cancel the subscription which brought it.

The display application is probably the most complicated part of the advice system; even apart from the need to include an HTML display engine, it may take three months or more to have all of its basic functionality in place.

The Advice Checker

The advice checker's job is to keep track of which advice is relevant and which advice is irrelevant. It isn't so much an application as a module used by the display application to evaluate guards and (depending on the updating model we choose) by a demon which updates the values of the guards when the display

application isn't open.

The difficulty of writing the advice checker is connected to the guard language. I expect we could use AppleScript as a guard language for prototype purposes, and have the checker working in a matter of a few weeks. However, AppleScript probably won't suffice for a shipping version; using it in guards seems to present too many security problems. Putting together a checker with a full language interpreter could take two to four months.

Associated with the checker is a mechanism for extracting basic facts from the system; in some sense, these basic relationships give a vocabulary to go with the advice checker's grammar. We'll want to have a plug-in interface to the basic facts, which will take about two weeks to produce; in addition we'll want to provide some basic facts, including:

- devices
- computer & CPU model
- amount of RAM
- directory and file info
- system components: applications, control panels, fonts, etc.
- version numbers
- gestalt values
- advice subscriptions
- advice stored in the system
- advice which has been triggered

The individual checkers will be simple, with most taking less than a day to produce.

The Advice Gatherer

This is the part which which will download advice from the net. Its heart will be an implementation of HTTP and SHTTP. I imagine that the HTTP can be up and running in about two month's work; while I'm less familiar with SHTTP, I suspect it will take less than one month more.

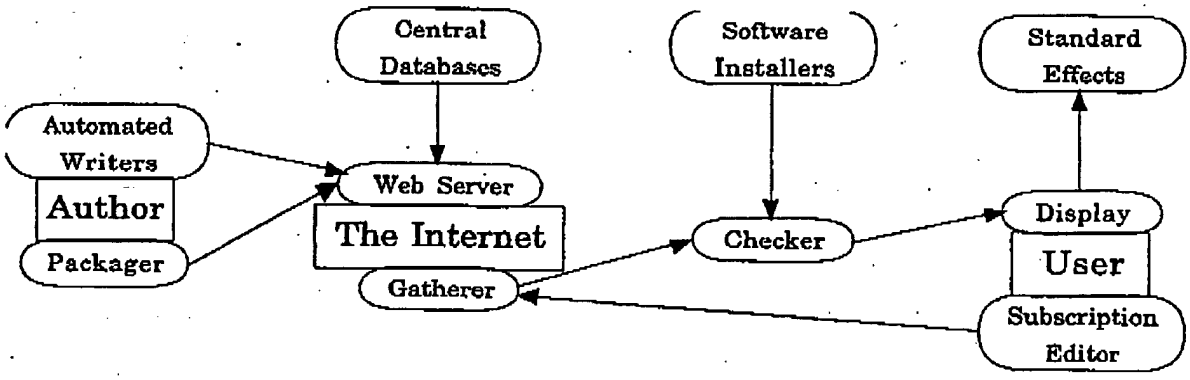
The Advice Packager

The packager is the basic authoring tool; it will work much like an email editor combined with the advice display application. (It may even be appropriate to sell the packager as an enhanced version of the client program, as with Netscape.) The basic interface is a text editor with separate fields for the header lines which the author can fill in. When the advice has been written, the packager fills in the computer-generated header lines.

In its basic form, the packager is a simple application, and should take less than a month to write. If we expand its definition to include editing HTML, it becomes a much more lengthy project.

The Peripheral Parts

While the parts above form the core of the advice system, there are several other parts we want to add to give the system a richer flavor.



Subscription Editor

The subscription editor is the interface through which the user can add and remove subscriptions. For each site the user subscribes to, the subscription editor will keep track of

- the site's URL